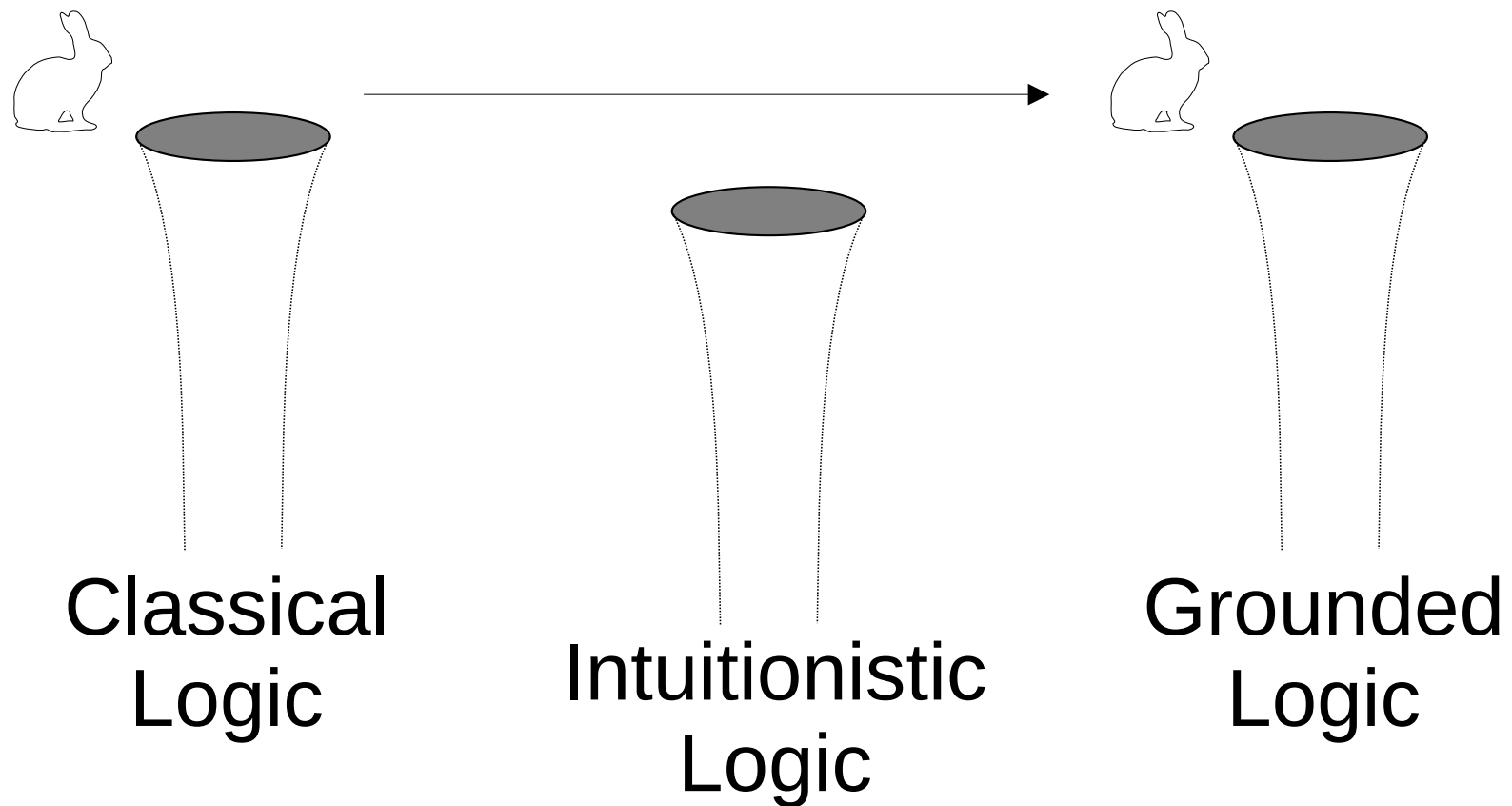


Have a thing?
Reasoning around recursion
with dynamic typing
in grounded arithmetic

Bryan Ford
EPFL – IC – DEDIS
bryan.ford@epfl.ch

MIT CSAIL – 26 March 2026

A Tale of Logical Rabbit Holes



Talk Outline

- **Motivation: can we have “nice things”?**
- Introduction: grounded arithmetic (GA)
- Habeas quid: enabling freedom of recursion
- Usability: can we do ordinary proofs in GA?
- Metatheory: what properties does GA have?
- Speculation: where might this lead us?

A few “nice things” to have

Practical:

1. Freedom of recursive definition
2. Freedom of mutual recursion

Metalogical:

1. Power + Consistency + Completeness
2. Logical Reflection

Can we have these “nice things”?

The “freedom of recursion” problem

Casual programming

YES



```
main() { printf("Hi!"); }  
main() { main(); }
```

permissionless

Classical formal logic

NO





*“Show me
your permit!”*

permissioned



Sane (non-local) mutual recursion

Module/package A:  Module/package B:
foo(...): ...bar(...)...  bar(...): ...foo(...)...

Casual programming

YES



no problem!

Formal logic

NO



tie a tight knot!



Example: Isabelle/HOL

Define mutually-recursive types, funs all at once

```
datatype tm = ... | tmIf fm tm tm | ...  
and fm = ... | fmEq tm tm | ...
```

Prove inductive theorems about them all at once

```
lemma to_trm_inj:  
  shows "tm_to_trm a = tm_to_trm b  $\implies$  a = b"  
  and "fm_to_trm p = fm_to_trm q  $\implies$  p = q"  
proof (induction a and p arbitrary: b and q)  
  ...
```

Example: Coq/Rocq

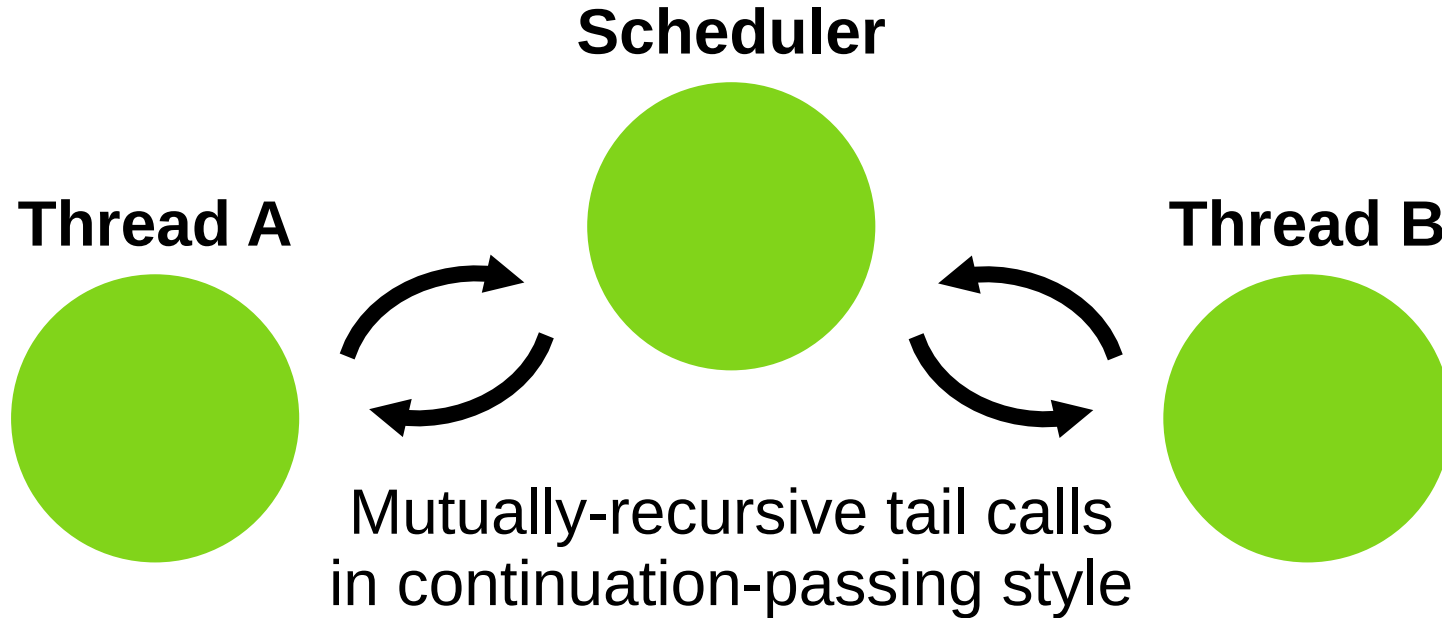
From the [current Rocq manual](#):

Mutually inductive types can be defined by including multiple *inductive_definitions*. The *idents* are simultaneously added to the global environment before the types of constructors are checked. Each *ident* can be used independently thereafter. However, the induction principles currently generated for such types are not useful. Use the **Scheme** command to generate useful induction principles. See [Mutually defined inductive types](#).

“Want mutual recursion? OK but first learn this new corpus of magic incantations!”

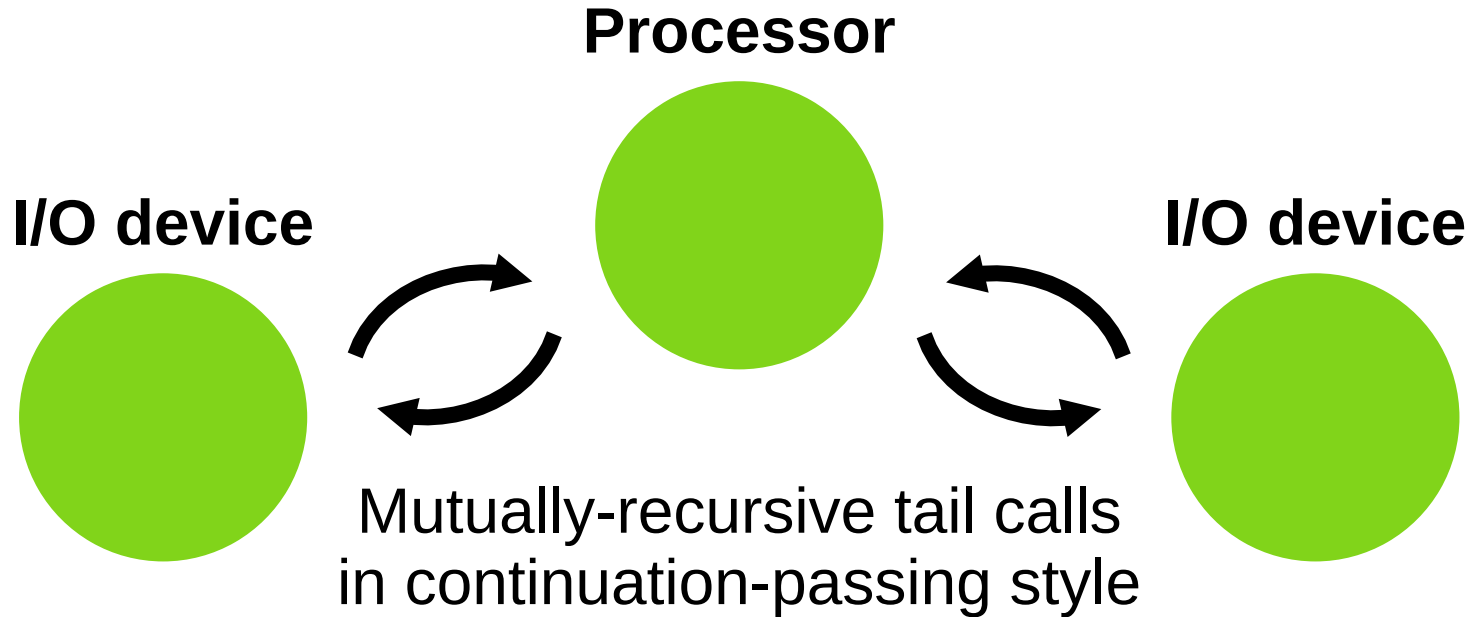
Why might we care?

Example: verifying multi-thread/process systems



Why might we care?

Example: modeling whole systems including I/O



Why might we care?

Example: verifying reentrant smart contracts

- Most famous “negative example”: **The DAO**
 - DAO code invokes attacker’s contract
 - Attacker’s contract re-invokes DAO for fun & profit
- Typical response: verify **absence** of reentrancy
 - “Sorry, reentrancy is a *nice thing you can’t have*”
- Couldn’t we verify the **safe use** of reentrancy?

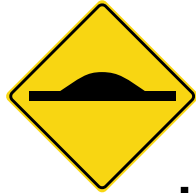
Static & dependent type systems

Useful to automate tedium, catch basic errors

But type systems always have **limitations**...

Casual programming

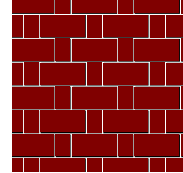
Speed bump



Escape/workaround via
dynamic checks, casts

Formal logic, verification

Brick wall



No escape or recourse:
major redesign/upgrade

Metatheory: wouldn't it be “nice” if...

Truth and provability
were aligned?

- Everything **provable** should be **true**
- Everything **true** should be **provable**

Classical logic

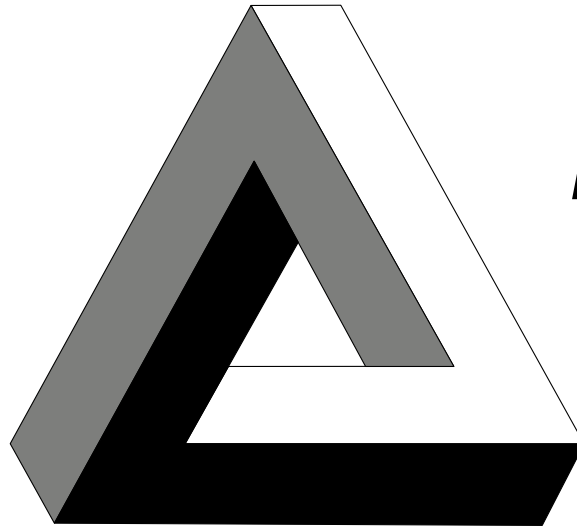
“Impossible!”



Gödel's incompleteness theorem

“Gödel’s trilemma”

Expressive Power
arithmetic + computation



*In “any formal system”
choose only two!*

Consistency
“anything provable is true”

Completeness
“anything true is provable”

“Can we do *this* interesting thing”?

Casual programming



Classical logic

“Impossible!”



Gödel's incompleteness theorem

Talk Outline

- Motivation: can we have “nice things”?
- **Introduction: grounded arithmetic (GA)**
- Habeas quid: enabling freedom of recursion
- Usability: can we do ordinary proofs in GA?
- Metatheory: what properties does GA have?
- Speculation: where might this lead us?

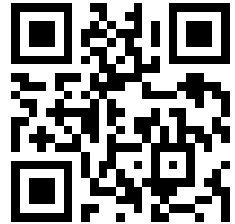
Broad kinda-readable overview

Reasoning Around Paradox with Grounded Deduction

Bryan Ford

[arXiv preprint 2409.08243](https://arxiv.org/abs/2409.08243)

September 12, 2024 (first version)



Abstract:

How can we reason around logical paradoxes without falling into them? This paper introduces grounded deduction or GD, a Kripke-inspired approach to first-order logic and arithmetic that is neither classical nor intuitionistic, but nevertheless appears both pragmatically usable and intuitively justifiable. GD permits the direct expression of unrestricted recursive definitions -- including paradoxical ones such as 'L := not L' -- while adding dynamic typing premises to certain inference rules so that such paradoxes do not lead to inconsistency. This paper constitutes a preliminary development and investigation of grounded deduction, to be extended with further elaboration and deeper analysis of its intriguing properties.

Grounded Arithmetic (GA): Goal

Define a *usable, working* paracomplete system

- Subject matter: natural numbers, computation
- Propositional $\neg \wedge \vee \rightarrow \leftrightarrow$ and quantifiers $\forall \exists$
- Inference rules: infer *terminating* computations
- *Safely avoid* (“reason around”) nontermination: ungrounded statements, functions yielding \perp

Grounded Arithmetic: non-goals

For now, GA is *not* immediately concerned with:

- Proving *nontermination*
- Real numbers, transfinite sets
- Non-computable mathematical objects
- Proving that paradoxes are paradoxical

Let's walk before trying to run!

Talk Outline

- Motivation: can we have “nice things”?
- Introduction: grounded arithmetic (GA)
- **Habeas quid: enabling freedom of recursion**
- Usability: can we do ordinary proofs in GA?
- Metatheory: what properties does GA have?
- Speculation: where might this lead us?

Unconstrained recursion = “trouble”

The Liar Paradox:

“This sentence is false”

$$L \equiv \neg L$$

Circular aka recursive definitions 

Curry’s Paradox:

$$C \equiv C \rightarrow P$$

“If this sentence is true then pigs fly”

A (new?) reasoning principle

habeas quid

We must *have a thing* in order to use it.

“Python-style dynamic type checking”

Static versus dynamic typing

Classical deduction

Syntax is ≥ 2 -sorted

- *Formulas* denote truth
- *Terms* denote objects

LEM: any valid *formula*
is either true or false

Grounded deduction

Syntax is 1-sorted

- Everything is a *term*

Assume nothing about
what a term denotes

- Only what is *proven*

Boolean as a “dynamic type”

A term is boolean in grounded deduction *only* if it has been previously *proven* either true or false

$$a \text{ bool} \equiv a \vee \neg a \text{ true}$$

New ***habeas quid*** obligations require proving we “have a (boolean) thing” before certain inferences

Defusing paradoxes by example

Why Habeas quid avoids inconsistency due to:

- The Liar paradox (negation)
- Curry's paradox (implication)
- Yablo's paradox (quantification)
- Berry's paradox (representation)

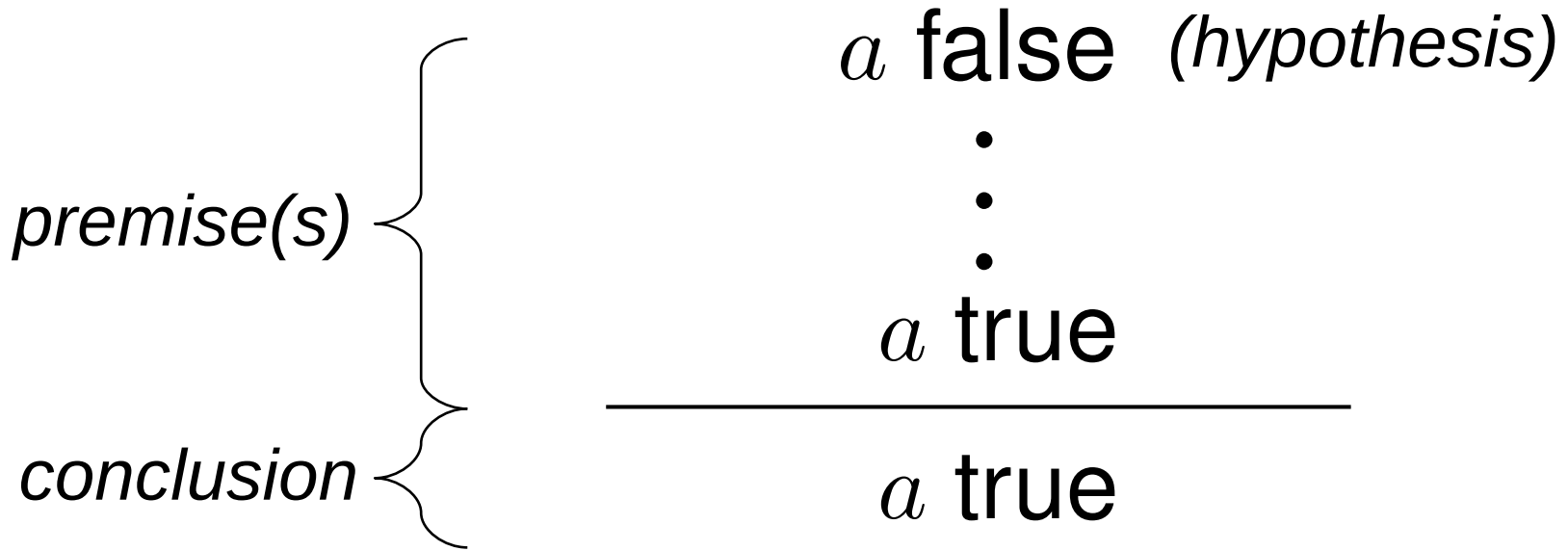
The Liar Paradox

“This sentence is false”

$$L \equiv \neg L$$

Natural deduction – inference rules

Proof by contradiction:



The Liar Paradox in Classical Logic

“This sentence is false”

$$L \equiv \neg L$$

L false (hypothesis)

L true (hypothesis)

$\neg L$ false (L 's definition)

$\neg L$ true (L 's definition)

L true (negation)

L false (negation)

L true (conclusion)

L false (conclusion)

The Liar Paradox in Classical Logic

“This sentence is false”

$$L \equiv \neg L$$

Classical deduction

L: “Not allowed!”

*Recursion must
be justified*



“Show me your permit!”

Grounded Logical Negation

Let's describe negation in terms of true and false:

$$\frac{a \text{ true}}{\neg a \text{ false}} \neg I1$$

$$\frac{a \text{ false}}{\neg a \text{ true}} \neg I2$$

$$\frac{\neg a \text{ false}}{a \text{ true}} \neg E1$$

$$\frac{\neg a \text{ true}}{a \text{ false}} \neg E2$$

Double-negation introduction and elimination hold:

$$\frac{a \text{ true}}{\neg\neg a \text{ true}} \neg\neg I$$

$$\frac{\neg\neg a \text{ true}}{a \text{ true}} \neg\neg E$$

...*unlike* in intuitionistic logic. *But...*

~~Classical~~ proof by contradiction Grounded



a bool

habeas quid

a false

⋮

a true

a true

The Liar Paradox

“This sentence is false”

$$L \equiv \neg L$$

Classical deduction

“Not allowed!”

*Recursion must
be justified*

“Show me your permit!”



Grounded deduction

Valid recursive definition

But is L a (bool) thing?

- To use contradiction,
first prove L boolean

Grounded logical implication

GA defines implication as in classical logic:

$$a \rightarrow b \equiv \neg a \vee b$$

...again *unlike* intuitionistic logic

And similarly biconditional (if-and-only-if):

$$a \leftrightarrow b \equiv (a \rightarrow b) \wedge (b \rightarrow a)$$

Grounded logical implication

Introduction and elimination rules for implication:

NEW

a true

a bool

:

b true

(modus ponens)

$$\frac{\text{habeas quid } a \text{ bool } \quad b \text{ true}}{a \rightarrow b \text{ true}} \rightarrow I$$

$$\frac{a \rightarrow b \text{ true} \quad a \text{ true}}{b \text{ true}} \rightarrow E$$

Curry's Paradox

“If this sentence is true then pigs fly” $C \equiv C \rightarrow P$

Classical/intuitionistic

Illegal circular definition!

If allowed, inconsistency

- With only $\rightarrow I$ and $\rightarrow E$
(no classical LEM)

Grounded deduction

Valid recursive definition

But is C boolean?

- To introduce $C \rightarrow P$,
first prove C boolean

Yablo's paradox “without self-reference”

Suppose we have an infinite series of statements:

- 1) All the following statements are false.
- 2) All the following statements are false.
- 3) ...

Are any of these statements true (or false)?

Yablo's paradox

Directly expressible in GA using quantifiers:

$$Y_i \equiv \forall_{j>i}(\neg Y_j)$$

Taking indexed Y_i as function $Y(i)$ with argument i

All are ungrounded/nonterminating: Y_i denotes \perp

Berry paradox

What (natural) number does this phrase denote?

“the least number not nameable in under ten words”

We can formalize this phrase in GA via reflection:
a computation that *tries to find* such a number,
but on the way recursively re-invokes itself (\perp)

Talk Outline

- Motivation: can we have “nice things”?
- Introduction: grounded arithmetic (GA)
- Habeas quid: enabling freedom of recursion
- **Usability: can we do ordinary proofs in GA?**
- Metatheory: what properties does GA have?
- Speculation: where might this lead us?

Can we do *useful* reasoning in GA?

Yes.

- Simple example: “even” function (next)
- Mutual recursion example: Cantor pairing
- In-progress: GA in the Isabelle proof assistant
- Metatheory: any general-recursive computation

How useful? Only time, experience will tell...

Satisfying *habeas quid* obligations

Example: $\text{even}(a) \equiv \begin{cases} \text{true} & | a = 0 \\ \neg \text{even}(a - 1) & | a > 0 \end{cases}$

$$\underbrace{x \text{ nat} \quad \text{even}(x) \text{ bool}}_{\vdots}$$

Habeas quid
proof by
induction:

$$\frac{\begin{array}{ll} (\text{true}) \text{ bool} & \neg \text{even}(x) \text{ bool} \\ \text{even}(0) \text{ bool} & \text{even}(x + 1) \text{ bool} \end{array}}{\forall x \text{ even}(x) \text{ bool}}$$

Habeas quid and partial functions

What if we invoke `even(i)` with an *integer* i ?

- $i \geq 0$: terminates, the above proof still works
- $i < 0$: fails to terminate, no *habeas quid* proof

The *existence* of nonterminating inputs does not hinder the function's *use* on terminating inputs

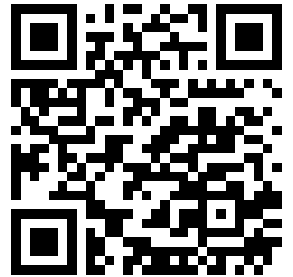
- Partial functions are not a problem

More first-hand use experience

Formalizing Grounded Arithmetic atop Isabelle/Pure

Sascha Kehrli

B.Sc. thesis advised by Bryan Ford and Roger Wattenhofer
September 30, 2025



Abstract:

This thesis presents a foundational formalization of Grounded Arithmetic (GA), a first-order arithmetic based on the principles of Grounded Deduction (GD), directly within the Isabelle/Pure framework. Unlike classical and constructive logics, which impose strict termination requirements on definitions to preserve consistency, GD admits arbitrary recursion at the definitional level. To remain consistent, GA weakens other inference rules, many of which demand explicit *habeas quid* termination proofs of subexpressions as premises. The goal of this thesis is to investigate the feasibility of GA as a practical basis for mathematical and computational reasoning by fully axiomatizing it in Pure.

Untangling mutual recursion

Classical and intuitionistic formal systems:
mutually-recursive data types, functions definable
only *all at once* with a mutual termination proof

Grounded arithmetic: no such limitation;
“freedom of mutual recursion”

Example: Isabelle/HOL (Classical)

Define mutually-recursive types, funs all at once

```
datatype tm = ... | tmIf fm tm tm | ...  
and fm = ... | fmEq tm tm | ...
```

Prove inductive theorems about them all at once

```
lemma to_trm_inj:  
  shows "tm_to_trm a = tm_to_trm b  $\implies$  a = b"  
  and "fm_to_trm p = fm_to_trm q  $\implies$  p = q"  
proof (induction a and p arbitrary: b and q)
```

...

Example: Cantor pairings in GA

Defining pairing and projection recursively:

```
cpair_def: "cpair x y := if y = 0 then div (x * S(x)) 2  
           else cpair x P(y) + x + y + 2"
```

```
cpx_def: "cpx x := if x = 0 then 0  
          else if cpx (P x) = 0 then S(cpy P(x))  
          else P(cpx (P x))" and
```

```
cpy_def: "cpy x := if cpx (P x) = 0 then 0  
          else S(cpy (P x))"
```

**mutual
recursion**

The diagram consists of two orange ovals. The first oval is around the expression 'S(cpy P(x))' in the 'cpx_def' line. The second oval is around the expression 'cpx (P x)' in the 'cpy_def' line. Two orange arrows originate from these ovals: one points from the 'S(cpy P(x))' oval to the 'cpx (P x)' oval, and the other points from the 'cpx (P x)' oval to the 'S(cpy P(x))' oval, forming a cycle that illustrates mutual recursion.

Kehrli, "Formalizing Grounded Arithmetic atop Isabelle/Pure"

We can have “freedom of recursion”

Unconstrained (even “paradoxical”) recursion

- Prove termination separately, later/elsewhere, only for inputs you care about – or not at all

Sane non-local mutual recursion

- Mutually recursive objects need not be defined, or proven terminating, “all at once” together

Sounds “nice” but...

How can we *know* this actually works?

Was inconsistency just “swept under the carpet”?

Historical lesson: skepticism is warranted!

Talk Outline

- Motivation: can we have “nice things”?
- Introduction: grounded arithmetic (GA)
- Habeas quid: enabling freedom of recursion
- **Metatheory: what properties does GA have?**
- Speculation: where might this lead us?

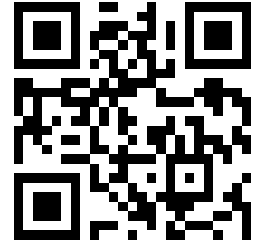
Grounded arithmetic metatheory

Have a thing? Reasoning around recursion with dynamic typing in grounded arithmetic

Elliot Bobrow, Bryan Ford, and Stefan Milenković

[arXiv preprint 2510.25369](https://arxiv.org/abs/2510.25369)

October 29, 2025 (first version)



Abstract:

Neither the classical nor intuitionistic logic traditions are perfectly-aligned with the purpose of reasoning about computation, in that neither tradition can permit unconstrained recursive definitions without inconsistency: recursive logical definitions must normally be proven terminating before admission and use. We introduce grounded arithmetic or GA, a formal-reasoning foundation allowing direct expression of arbitrary recursive definitions. GA adjusts traditional inference rules so that terms that express nonterminating computations harmlessly denote no semantic value (i.e., \perp) instead of yielding inconsistency. Recursive functions may be proven terminating in GA essentially by “dynamically typing” terms, or equivalently, symbolically reverse-executing the computations they denote via GA's inference rules. Once recursive functions have been proven terminating, logical reasoning about their results reduce to the familiar classical rules. A mechanically-checked formal development of basic grounded arithmetic or BGA – a minimal kernel for GA – shows that BGA simultaneously exhibits the useful metalogical properties of being (a) semantically and syntactically consistent, (b) semantically complete, and (c) sufficiently powerful to express and prove the termination of arbitrary closed Turing-complete computations. This combination is impossible in classical logic due to Gödel's incompleteness theorems, but our results do not contradict Gödel's theorems because BGA is paraconsistent, not classical. Leveraging BGA's power of computation and reflection, we find that grounded logical operators including quantifiers are definable as non-primitive computations in BGA, despite not being included as primitives.

Basic Grounded Arithmetic (BGA)

Minimalistic computational foundation for GA

- Term syntax: 0, successor, predecessor, if-zero-else, 2-argument function call

$$t \equiv v \mid 0 \mid S t \mid P t \mid t \ 0? \ t : t \mid d(t, t)$$

- Formula syntax: equal (=), unequal (\neq)

$$f \equiv t = t \mid t \neq t$$

- Fixed list of 2-argument recursive definitions

Mechanically-verified metatheory

Basic Grounded Arithmetic (BGA)



Logic syntax, proof theory, model theory, recursion theory



Metalogic: Isabelle/HOL (Higher-Order Logic)



Zermelo-Fraenkel set theory with Choice (ZFC)

Inference/proof rules of BGA

(Just) sufficient to prove results of computations

$$\begin{array}{cccccc}
 \frac{s(\vec{x}) \equiv d(\vec{x}) \quad p(d(\vec{a}), \dots)}{p(s(\vec{a}), \dots)} \equiv I & \frac{a = b}{b = a} = S & \frac{a = b \quad p(a, \dots)}{p(b, \dots)} = E & \frac{a \neq b}{b \neq a} \neq S & \frac{}{0 \in \mathbb{N}} 0I & \frac{a = b}{S(a) = S(b)} S=I \\
 \frac{a \neq b}{S(a) \neq S(b)} S \neq I & \frac{a \in \mathbb{N}}{S(a) \neq 0} S \neq 0I & \frac{a = S(b)}{P(a) = b} PI & \frac{c = 0 \quad a \in \mathbb{N}}{(c \neq 0 \rightarrow a : b) = a} ?I1 & \frac{c \neq 0 \quad b \in \mathbb{N}}{(c \neq 0 \rightarrow a : b) = b} ?I2
 \end{array}$$

No primitive induction! (like Robinson arithmetic)

Semantic model of BGA

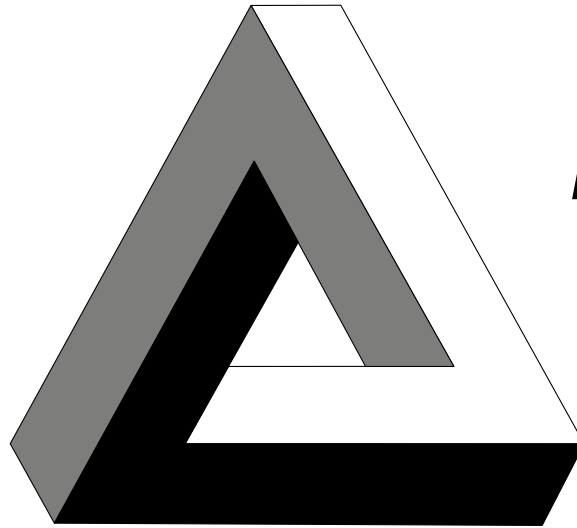
Used an operational semantic reduction model...

$$\begin{array}{c}
 \text{Term reductions} \\
 \frac{A(\mathbf{v}_j) = n}{\mathbf{v}_j \Downarrow n} \quad \frac{}{0 \Downarrow 0} \quad \frac{a \Downarrow n}{S(a) \Downarrow n+1} \quad \frac{a \Downarrow n+1}{P(a) \Downarrow n} \quad \frac{c \Downarrow 0 \quad a \Downarrow n}{c \text{ 0? } a : b \Downarrow n} \quad \frac{c \Downarrow 1+m \quad b \Downarrow n}{c \text{ 0? } a : b \Downarrow n} \quad \frac{a_0 \Downarrow n_0 \quad a_1 \Downarrow n_1 \quad D_i \langle \underline{n_0}, \underline{n_1} \rangle \Downarrow m}{\mathbf{d}_i(a_0, a_1) \Downarrow m} \\
 \\
 \text{Formula reductions} \\
 \frac{a \Downarrow n \quad b \Downarrow n}{a = b \Downarrow 1} \quad \frac{a \Downarrow n \quad b \Downarrow m \quad n \neq m}{a = b \Downarrow 0} \quad \frac{a \Downarrow n \quad b \Downarrow n}{a \neq b \Downarrow 0} \quad \frac{a \Downarrow n \quad b \Downarrow m \quad n \neq m}{a \neq b \Downarrow 1}
 \end{array}$$

Denotational semantic model should also work

“Gödel’s trilemma”

Expressive Power
arithmetic + computation



*In ~~any~~ formal system”
choose only two!*

Consistency
“anything provable is true”

Completeness
“anything true is provable”

“Gödel’s trilemma”

Expressive Power
arithmetic + computation



*In any
classical or intuitionistic
formal system,
choose two!*

Consistency
“anything provable is true”

Completeness
“anything true is provable”

In Basic Grounded Arithmetic (BGA)

~35K line Isabelle/HOL metatheory shows:

- BGA has **power to express any computation**
 - Via unconstrained recursive definitions
- BGA is semantically & syntactically **consistent**
 - With respect to an operational-semantic model
- BGA is semantically **complete**
 - Anything true in its semantic model is provable

Semantic and syntactic consistency

LEMMA 4.5 (TRUTH PRESERVATION). *The inference rules of BGA in Table 2c are truth preserving.*

THEOREM 4.6 (SEMANTIC CONSISTENCY). *Every theorem of BGA is true in the operational-semantic model of BGA.*

THEOREM 4.7 (SYNTACTIC CONSISTENCY). *There are no terms t_1, t_2 such that both ' $t_1 = t_2$ ' and ' $t_1 \neq t_2$ ' are theorems of BGA.*

Semantic completeness

LEMMA 4.8. *For any well-formed term t that reduces to a natural number r in s steps under all assignments via BGA's operational semantics, the judgment ' $\emptyset \vdash t = \underline{r}$ ' is provable in BGA.*

THEOREM 4.9 (SEMANTIC COMPLETENESS). *Every semantically true formula in BGA is provable using the rules in Table 2c.*

What about syntactic completeness?

Standard (classical) definition:

for every w.f. formula f , either f or $\neg f$ is provable.

Impossible in *any* consistent paracomplete system

- Otherwise it wouldn't be paracomplete!

But unlike in classical logic,

syntactic and semantic completeness are “untied”

Representation of computation

THEOREM 4.12 (GENERAL RECURSIVE FUNCTIONS). *For any 1-argument general-recursive function $f(x)$ defined by primitive-recursive step function $f_s(x, y)$, there is a definition list D yielding a BGA instance B_D that represents $f(x)$.*

THEOREM 4.14 (UNIVERSAL RECURSION). *The universal BGA instance B_U represents every general-recursive function.*

Why is this combination important?

Intuitive essence:

- Consistency: true and false are distinguishable
- Completeness: truth and proofs well-aligned
 - Can “traverse” freely between truth and provability
- Computation: evidence of power, generality

Together: strong Gödel-style reflective reasoning

What about logical connectives?

Not primitive in BGA, but definable computations

THEOREM 4.15 (ARITHMETIZATION). *Functions defining the syntax and proof rules of BGA are represented in BGA instance B_U .*

...using Gödel's arithmetic reflection techniques

What about logical connectives?

Not primitive in BGA, but definable computations

THEOREM 4.15 (ARITHMETIZATION). *Functions defining the syntax and proof rules of BGA are represented in BGA instance B_U .*

THEOREM 4.16 (GROUNDED NEGATION). *For any BGA for-*

THEOREM 4.17 (GROUNDED DISJUNCTION). *For any BGA formulas f_1 and f_2 , a formula ' $f_1 \vee f_2$ ' is constructible in BGA instance B_U such that ' $f_1 \vee f_2$ ' is true iff either f_1 or f_2 are true, and such that ' $f_1 \vee f_2$ ' is false iff both f_1 and f_2 are false.*

What about quantification?

Unlike in classical Peano arithmetic or set theory, quantifiers in grounded arithmetic are computable ...and can even represent mathematical induction

THEOREM 4.18 (INDUCTIVE UNIVERSAL QUANTIFICATION).
For any BGA predicate $p\langle x \rangle$ with one free variable x , a formula $\forall x p\langle x \rangle$ is constructible in BGA instance B_U that: (a) evaluates to true if there are BGA proofs of $\emptyset \vdash p\langle 0 \rangle$ and $\{x \ N, p\langle x \rangle\} \vdash p\langle Sx \rangle$, and (b) evaluates to false if there is a BGA ~~proof~~ of $\emptyset \vdash p\langle \underline{n} \rangle$ for some natural number n .

refutation

Upshot: predicate logic “for free”

All logical connectives definable as computations
...using Gödel-style reflection “constructively”

THEOREM 4.20 (GROUNDED PREDICATE LOGIC). *All of the grounded predicate logic operators in Section 3 are represented as computations in the universal BGA instance B_U .*

...to prove stuff *possible* rather than *impossible*

Summary: “nice things” we *can* have

Practical:

1. Unconstrained recursion
2. Sane mutual recursion

Metalogical:

1. Power + Consistency + Completeness
2. Reflection

...with grounded but not classical reasoning

Talk Outline

- Motivation: can we have “nice things”?
- Introduction: grounded arithmetic (GA)
- Habeas quid: enabling freedom of recursion
- Metatheory: what properties does GA have?
- **Speculation: where might this lead us?**

Work in progress: *more speculative*

Practical usability development

- Beyond arithmetic: rich data types, real, sets
- Can we have “point-like” grounded reals?

Further metatheory development

- How far does power of grounded reasoning go?
- How does it compare with PA, ZF set theory?

Reflection à la Gödel

Let us try to prove Gödel's theorems about GA.

- Gödel coding and reflection on provability: YES
- Diagonalization (aka fixed-point) lemma: NO
 - Must weaken with *habeas quid* precondition

What does this mean for Gödel's theorems?

Reflection à la Gödel

Gödel 1st “there is an undecidable sentence”

- **YES:** not Gödel’s way, but more trivially via Liar

Gödel 2nd “a system can’t prove itself consistent”

- **NO:** proof doesn’t work, no apparent fix
 - For same reason Liar paradox is defused

Bug or feature? *Can GA prove itself consistent?*

Proving paradoxes paradoxical

We might like to be able to say, and prove, that “the Liar sentence L is Not True (and Not False)”

- Without slipping down the slope of “revenge”



Did Gödel already show us a solution? *Reflection*

- “ L is Not True” $\equiv \neg \mathbf{Prv}(\ulcorner L \urcorner)$
- “ L is Not False” $\equiv \neg \mathbf{Prv}(\ulcorner \neg L \urcorner)$
- “ L is Ungrounded” $\equiv \neg \mathbf{Prv}(\ulcorner L \urcorner) \wedge \neg \mathbf{Prv}(\ulcorner \neg L \urcorner)$

Paradoxes and paradoxes-in-hiding

What if we fully “drink the grounded kool-aid” and view classical results from grounded perspective?

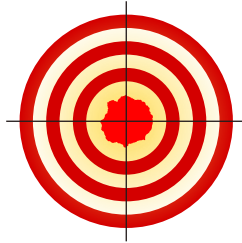
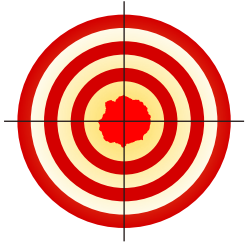
“Interesting things” happen to:

- Halting problem
- Cantor’s theorem
- Banach-Tarski paradox

The Halting Problem

Under a reasonable grounded perspective on what constitutes a computational “problem”...

- Must have a well-defined *target* or goal-posts



Search Problem

Sorting Problem

Halting Problem

The Halting Problem

Under a reasonable grounded perspective on what constitutes a computational “problem”...

- Must have a well-defined *target* or goal-posts
- Halting problem *statement* is non-terminating!
 - Classical reasoning: pretends it is well-defined
 - Grounded reasoning: no problem to be solved – *habeas quid* barrier to proving it is even a “problem”

Cantor's ~~theorem~~ "paradox in hiding"

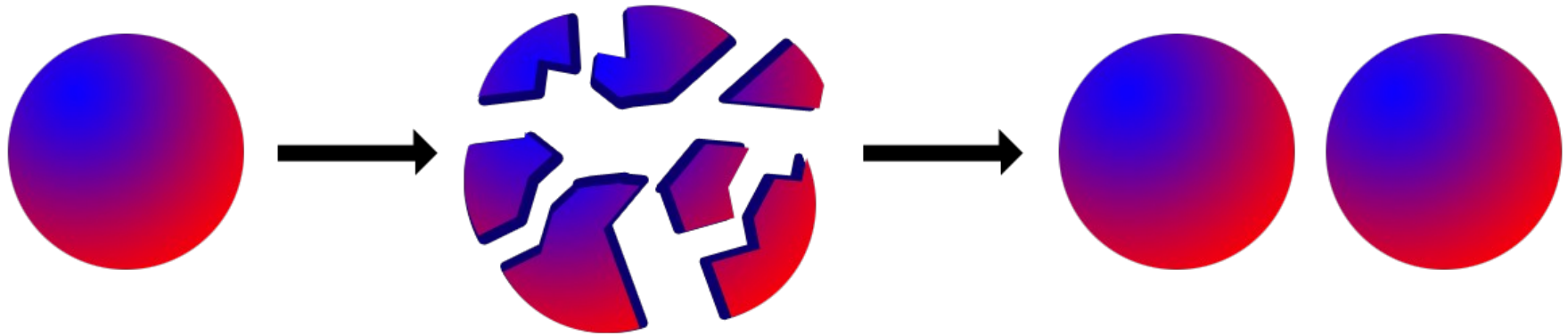
Supposing real numbers have an enumeration E , construct a new real number C not listed in E

- Grounded: reals must "decide" their "position"
 - Cantor's real "searches for itself" among all reals
 - "Finds itself" only given verifiable *proof* C is a real
 - C re-invokes itself: *habeas quid* violation, no proof

Grounded reals enumerable; cardinals collapse

Placing the blame for Banach-Tarski

In ZFC, we can reassemble 1 unit ball into 2 balls



[image credit: [cognitive coitus](#)]

Placing the blame for Banach-Tarski

In ZFC, we can reassemble 1 unit ball into 2 balls

Proof depends on Axiom of Choice (AC)

- Triggered intense controversy over AC

But: proof *also* depends on Cantor's Theorem

- Uncountable set \setminus countable set \neq empty

Was the “wrong suspect” historically on trial?



Logic vs logic “critiques”



Classical vs grounded:

“Impoverished!”

- Weakened inference
- Excluded middle lost
- Cantor’s paradise lost
- Non-computable lost

Grounded vs classical:

“Hallucinatory!”

- Weakened recursion
- Nonsense as “truth”
- Conjures phantasms
- Lost in a mirror maze

Conclusion: further reading

“Reasoning around paradox
with grounded deduction”

“Have a thing?
Reasoning around recursion...”

“Formalizing grounded arithmetic
atop Isabelle/Pure”

